High-performance deduction for verification: Synthetic benchmarks in the theory of arrays

Maria Paola Bonacina, Dip. Informatica, Università degli Studi di Verona, Italy

Joint work with:
Alessandro Armando, DIST, Università degli Studi di Genova, Italy
Silvio Ranise, LORIA & INRIA-Lorraine, Nancy, France
Michaël Rusinowitch, LORIA & INRIA-Lorraine, Nancy, France
Aditya Kumar Sehgal, Dept. of Computer Science, U. Iowa, USA

Motivation

SW verification/debugging requires reasoning with theories of data types, e.g., integer, real, arrays, lists, sets.

Problem: infinite domains.

Approach: model checking with theorem proving support, e.g., abstract-check-refine paradigm

[Blast : Henzinger et al.   POPL 2002]

[SLAM: Ball, Rajamani   POPL 2002]

[Simplify: Nelson et al. 1996]

Objective

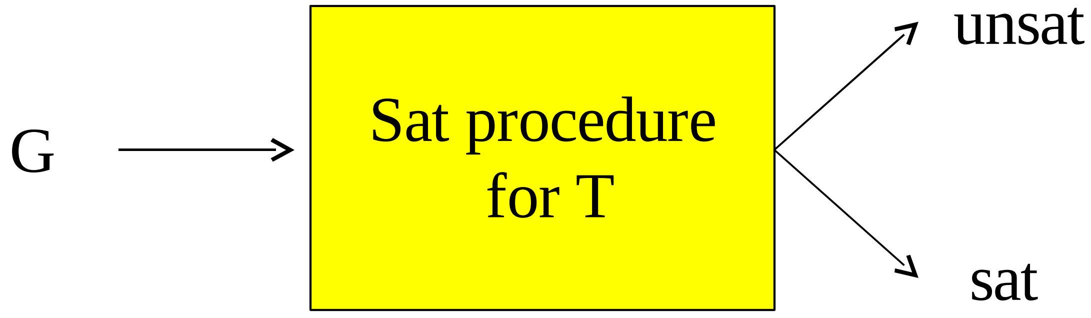
Offer better theorem proving support
to (model checking for) software analysis :

Better trade-off's between
termination and efficiency on one hand,
and correctness, completeness and expressive power
on the other

Satisfiability procedure

T : background theory (e.g., theory of arrays)

G : conjunction (set) of ground literals



Replace G by any quantifier-free formula : decision procedure

Common approach

- Design
- prove sound and complete
- and implement

a satisfiability procedure for each decidable theory of interest.

Basic ingredients:

- Defined symbols (in T) and free symbols
- Congruence closure (to handle equality and free symbols)
- Build axioms of T into congruence closure algorithm

Examples

Theory of lists :
congruence closure with axioms built-in
[Nelson, Oppen 1980]

Theory of arrays :
congruence closure with pre-processing wrt. axioms and partial equations (i.e., equalities that say that two arrays are equal except at certain indices)
[Stump et al., LICS 2001]

Issues

- Combination of theories/procedures
- Completeness proofs
- Implementation

Combination

Most problems involve multiple theories:

combination of theories / procedures

Two congruence-closure based approaches:

[Nelson, Oppen 1979] [Shostak 1984]

that generated much scholarship:

[Cyrluk, Lincoln, Shankar CADE 1996]

[Harandi, Tinelli 1998]

[Kapur RTA 2000]

[Ruess, Shankar LICS 2001]

[Barrett et al. FroCoS 2002]

[Ganzinger CADE 2002]

Completeness proofs

Each new decision procedure needs its own proofs of soundness and completeness:

- Proofs for concrete procedures : complicated, ad hoc

[Shankar, Ruess LICS 2001]
[Stump et al. LICS 2001]

- Abstract frameworks : clarity, but gap wrt concrete procedures

[Bjorner Phd thesis 1998]
[Tiwari Phd thesis 2000]
[Bachmair, Tiwari, Vigneron JAR 2002]
[Ganzinger CADE 2002]

Implementation

Implement from scratch data structures and algorithms for each procedure in each context
(e.g., verification tool or proof assistant):

- Correctness of implementation?
- Flexibility ?
- SW reuse?

Outline

- A deduction-based approach to address these issues

- Theory of arrays : synthetic benchmarks
- Experimental results
- Discussion

Relation to deduction

Although satisfiability procedures may not be presented/perceived as deduction proper, they are built out of deduction:

Congruence closure, normalization, canonical forms, reasoning modulo a theory, T- unification …

Could deduction help ?

Theorem proving could help:

- Combination of theories: give union of the axiomatizations in input to the prover
- No need of ad hoc proofs for each procedure
- Reuse code of existing provers

Termination ?

$C = < I, \Sigma >$: theorem-proving strategy

I : refutationally complete inference system with superposition/ paramodulation, simplification, subsumption …

Σ: fair search plan

is a semi-decision procedure:

$T \cup G$

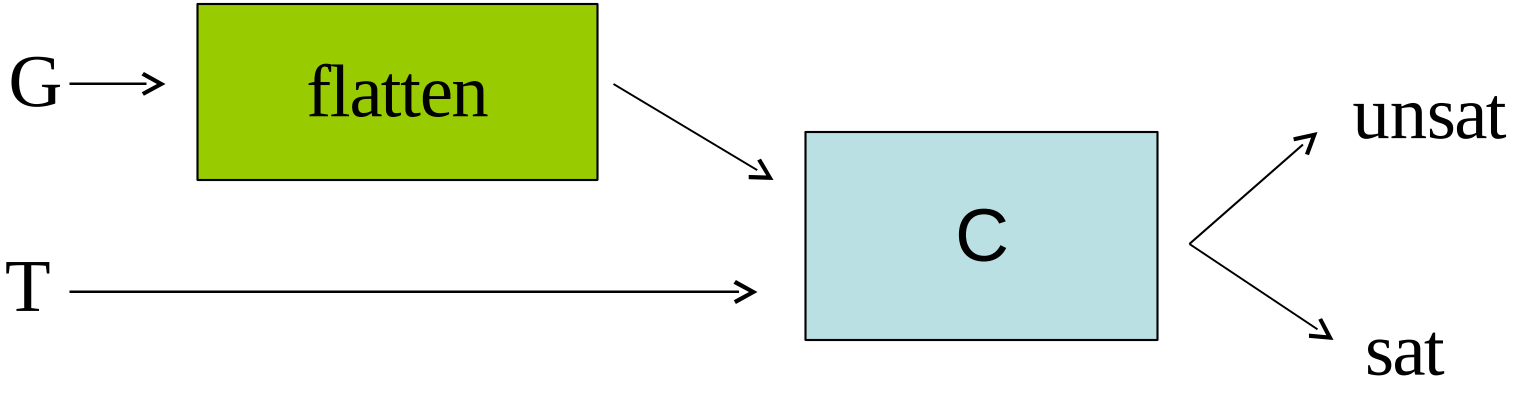
Yes, iff $T \cup G$ is unsatisfiable

?

Termination results

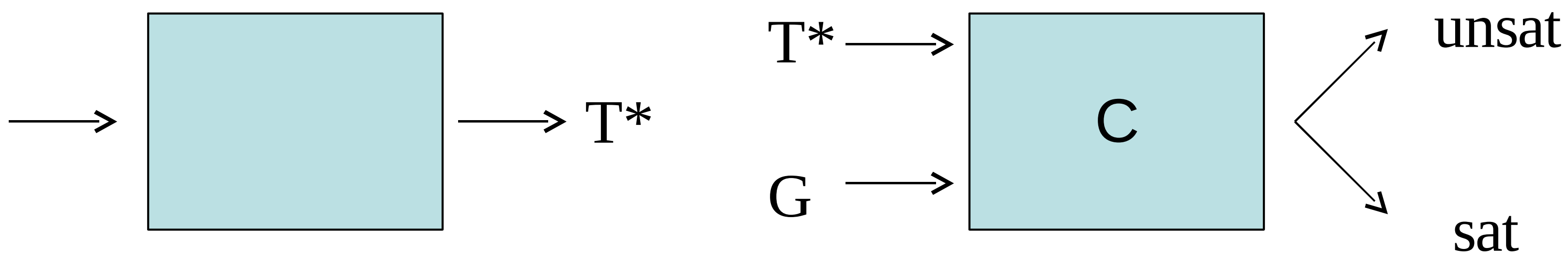
[Armando, Ranise, Rusinowitch, CSL 2001]

T: theory of arrays, lists, sets and combinations thereof



Arbitrary quantifier-free formulae : [Ranise UNIF 2002]

Another way to put it



Pure equational: T* canonical rewrite system

Horn equational: T* saturated ground-preserving
[Kounalis & Rusinowitch, CADE 1988]

FO special theories: e.g., T = T* for arrays [ARR, CSL 2001]

How about efficiency ?

A satisfiability procedure with T built-in is expected to be always much faster than a theorem prover with T in input !

Totally obvious ? Or worth investigating ?

- theory of arrays
- synthetic benchmarks (allow to assess scalability by experimental asymptotic analysis)
- comparison of E prover and CVC validity checker with theory of arrays built-in

Theory of arrays: the signature

$$\text{store} : \text{array} \times \text{index} \times \text{element} \rightarrow \text{array}$$

$$\text{select} : \text{array} \times \text{index} \rightarrow \text{element}$$

The presentation (T_1)

(1) $\forall$A, I, E. select (store (A, I, E), I) = E

(2) $\forall$A, I, J, E. I $\neq$ J $\Rightarrow$
select (store (A, I, E), J) = select (A, J)

(3) Extensionality: $\forall$A, B.
$\forall$I. select (A, I) = select (B, I)
$\Rightarrow$
A = B

Pre-processing extensionality

$$select(A, sk(A, B)) \neq select(B, sk(A, B)) \lor A = B$$

$$t \neq t'$$

$$select(t, sk(t, t')) \neq select(t', sk(t, t'))$$

Another presentation (T_2)

Keep (1) and (2) and replace extensionality (3) by:

(4) $\forall A, I.\ store(A, I, select(A, I)) = A$

(5) $\forall A, I, E, F.$
$store(store(A, I, E), I, F) = store(A, I, F)$

(6) $\forall A, I, J, E.\ I \neq J \Rightarrow$
$store(store(A, I, E), J, F) = store(store(A, J, F), I, E)$

T_1 entails (4) (5) (6)

Usage of presentations

- T_1 is saturated and application of C to $T_1 \cup G$ is guaranteed to terminate [ARR2001]: C acts as decision procedure

- T_2 is not saturated (saturation does not halt): C applied to $T_2 \cup G$ acts as semi-decision procedure

Synthetic benchmarks

- storecomm(N):

Storing values at distinct places
in an array is “commutative”

- swap(N):

Swapping pairs of elements in an array
in two different orders yields the same array

storecomm(N) : definition

$k_1 \ldots k_N$: N indices

D : set of 2-combinations over { 1 … N }

Indices must be distinct:

$$\bigwedge_{(p,\, q) \in D} k_p \neq k_q$$

$i_1 \ldots i_N$, $j_1 \ldots j_N$: two distinct permutations of 1 … N

$$\text{store}\,(\ldots(\,\text{store}\,(\,a,\, k_{i_1},\, e_{i_1}\,),\, \ldots k_{i_N},\, e_{i_N}\,)\,\ldots)$$

$$=$$

$$\text{store}\,(\ldots(\,\text{store}\,(\,a,\, k_{j_1},\, e_{j_1}\,),\, \ldots k_{j_N},\, e_{j_N}\,)\,\ldots)$$

storecomm(N) : schema

$$\bigwedge_{(p,\, q) \in D} k_p \neq k_q$$

$$\Rightarrow$$

$$\text{store}(\ldots(\text{store}(a, k_{i1}, e_{i1}), \ldots k_{iN}, e_{iN}) \ldots)$$
$$=$$
$$\text{store}(\ldots(\text{store}(a, k_{j1}, e_{j1}), \ldots k_{jN}, e_{jN}) \ldots)$$

swap(N) : definition

Recursively:

Base case: N = 2 elements:

L_2 = store (store (a, i_1, select (a, i_0)), i_0, select (a, i_1))

R_2 = store (store (a, i_0, select (a, i_1)), i_1, select (a, i_0))

$$L_2 = R_2$$

Recursive case: N = k+2 elements:

L_{k+2} = store (store (L_k, i_{k+1}, select (L_k, i_k)), i_k, select (L_k, i_{k+1}))

R_{k+2} = store (store (R_k, i_k , select (R_k, i_{k+1})), i_{k+1}, select (R_k, i_k))

$$L_{k+2} = R_{k+2}$$

Experiments

- Two tools: CVC validity checker and E theorem prover
- E: auto mode and user-selected strategy
- Comparison of asymptotic behavior of E and CVC as N grows

The CVC validity checker

[Aaron Stump, David L. Dill et al., Stanford U.]

Combines procedures à la Nelson-Oppen
(e.g., lists, arrays, records, real arithmetics ...)

Incorporates SAT solver (first GRASP then Chaff)
to handle arbitrary quantifier-free formulae

Theory of arrays: congruence closure based algorithm
[Stump et al., LICS 2001]

The E theorem prover

[Stephan Schulz, TU-München]

Inference system I : o-superposition/paramodulation, reflection, o-factoring, simplification, subsumption

Search plans Σ :

- given-clause loop with clause selection functions and only already-selected list inter-reduced
- term orderings: KBO and LPO
- literal selection functions

Strategies in experiments

- E-auto: automatic mode
- E-SOS: { problem in form $T \cup G$ }

  Clause selection: (SimulateSOS,RefinedWeight)

  Term ordering: LPO
- Precedence: select > store > sk > constants

First set of experiments on storecomm(N)

E takes presentation T_1 in input

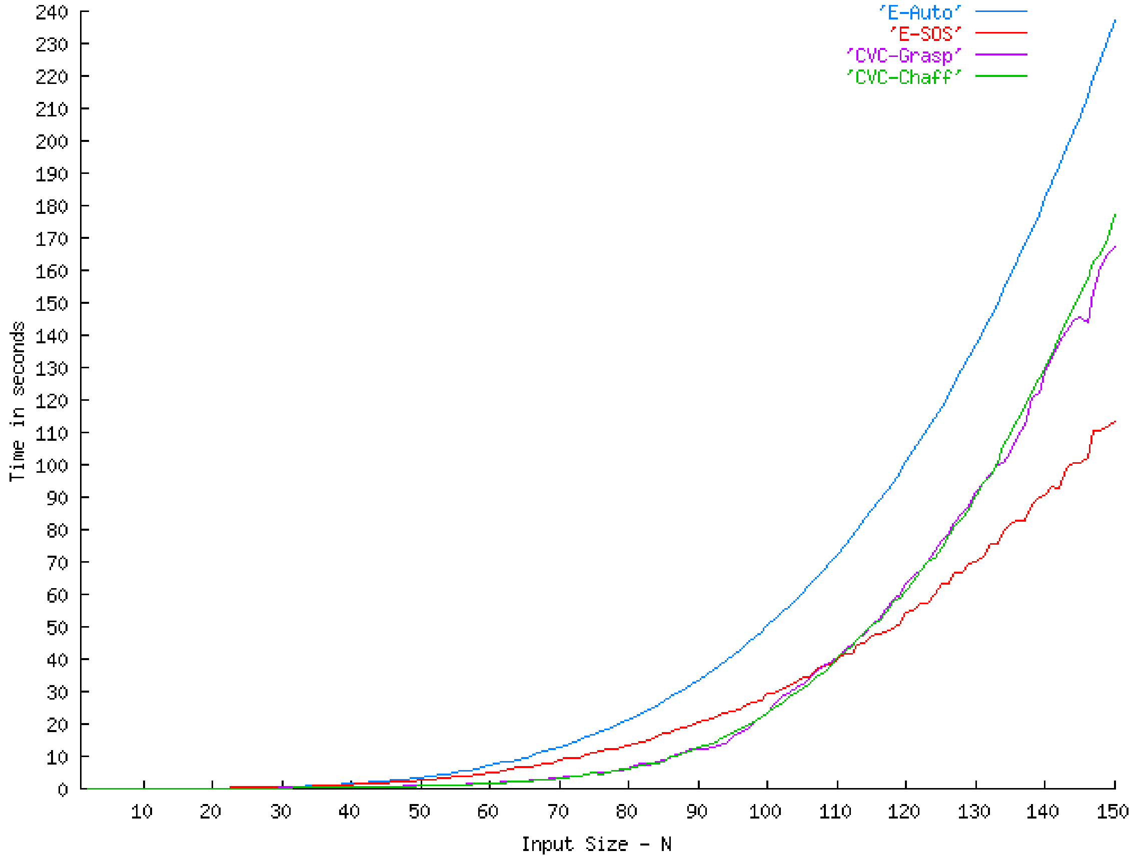
N ranges from 2 to 150

Sample 10 permutations: 45 instances for each value of N
Non-uniform sampling (favors permutations with local changes)

Performance for N is average over all generated instances for value N

Versions : E 0.62
CVC/GRASP Fall 2001, CVC/CHAFF January 2002

First set : storecomm(N)



Second set of experiments on storecomm(N)

E takes presentation T_1 in input

N ranges from 2 to 90

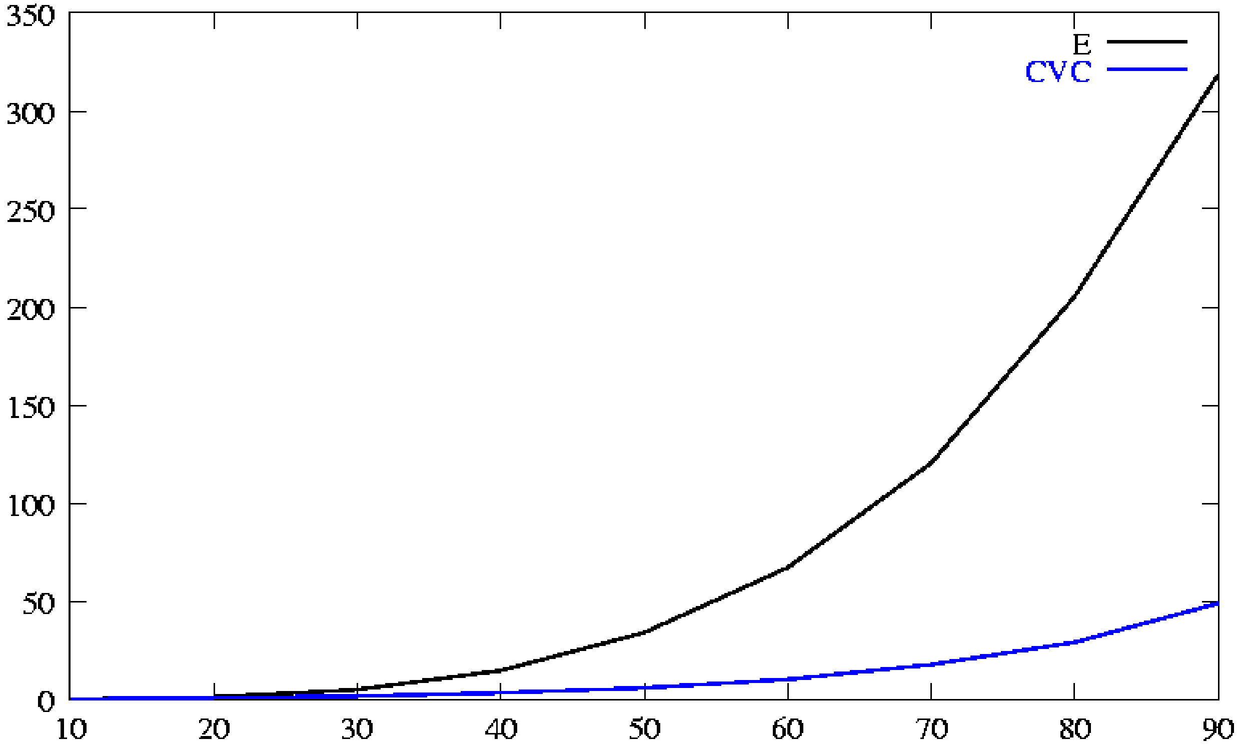
For each value of N pick one instance at random : no averages

Only E-auto, E-SOS did not help

Versions : E 0.62

CVC/CHAFF October 2002

Second set : storecomm(N)



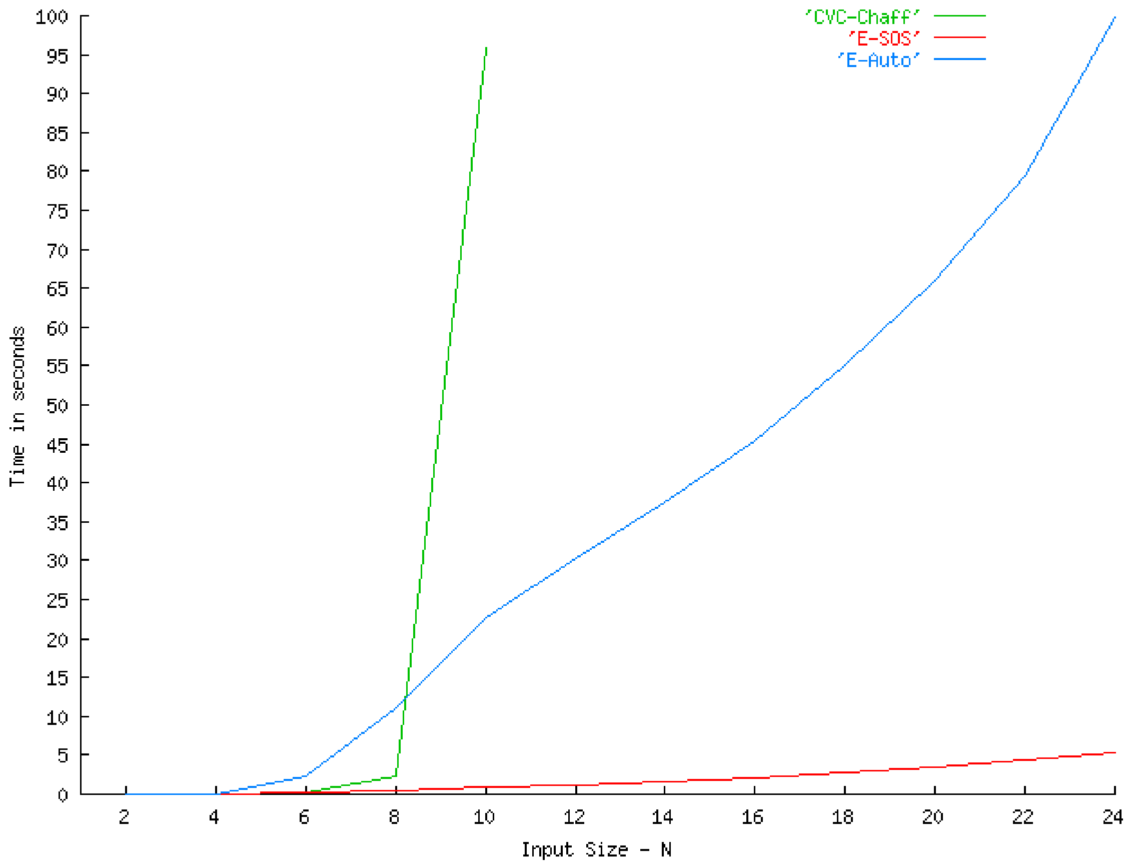
First set of experiments on swap (N)

Sample up to 16 permutations and 20 instances for each value of N
Non-uniform sampling (favors permutations with local changes)
Performance for N is average over all generated instances for value N

CVC: does up to N = 10, runs out of memory on
any instance of swap(12)
E with presentation T_1: same as above and slower
E with presentation T_2: succeeds also for $N \geq 12$

Versions : E 0.62
CVC/GRASP Fall 2001, CVC/CHAFF January 2002

First set : swap(N)



Second set of experiments on swap(N)

E takes presentation T_1 in input

For each value of N pick one instance at random : no averages

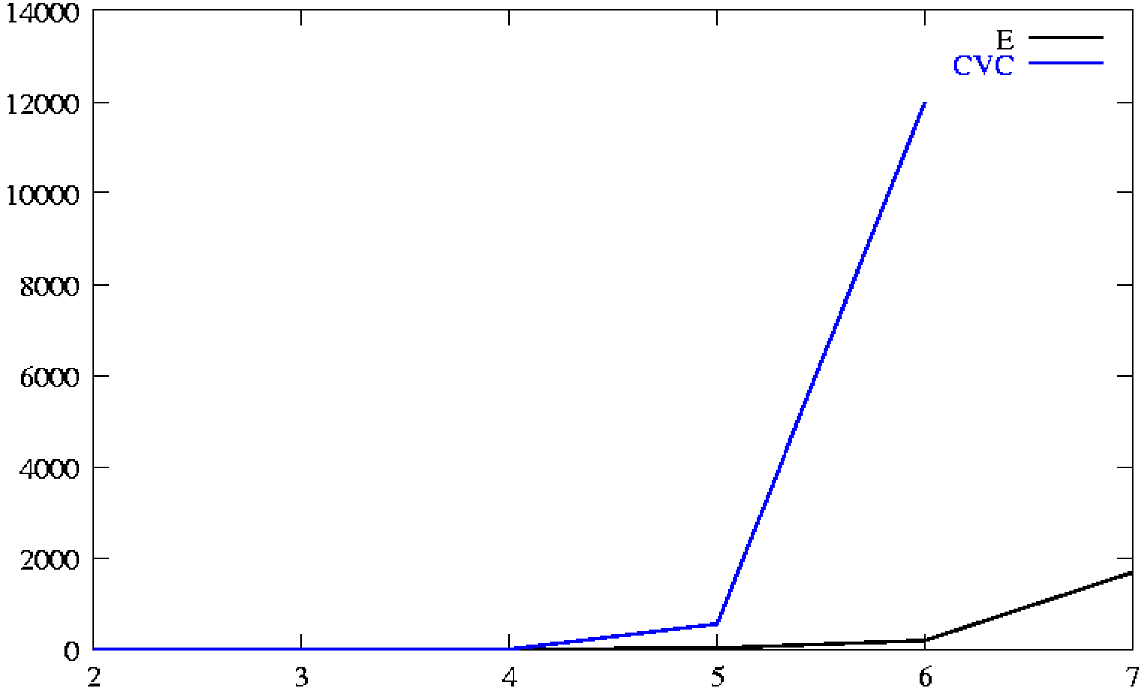
Only E-auto, E-SOS did not help

CVC : does only up to N = 6, E goes beyond

Versions : E 0.62

CVC/CHAFF October 2002

Second set : swap(N)



Discussion

- Deduction may help build better decision procedures
- More experiments: other synthetic benchmarks, theories, combinations, real-world problems
- Other provers, e.g., with more inter-reduction?
- ATP needs more work on auto mode and search plans (search, not blind saturation)
- Termination results for other theories?
- Complexity of specific strategies / theories

Broad picture : integration

- ATP based satisfiability procedures
- Integration with SAT : decision procedures
- Integration with automated model building:

counterexample generation

- Integration within debugging tools or proof assistants